基于闪存的外存储技术综述

魏巍　熊劲　蒋德钧

摘 要 闪存（Flash）在访问延迟、传输带宽、价格方面介于 DRAM 和磁盘之间，而且密度比 DRAM 和磁盘高，能耗比它们低，因此近几年基于闪存的外存储技术成为研究的热点。本文总结了目前基于闪存的外存储技术，重点分析了两类软件体系结构，以及与其相关的研究问题和关键技术。并且基于对研究现状的分析，我们认为未来的研究方向是如何充分发挥闪存存储器的最大价值。

1　引言

闪存是一种电可擦除可编程只读非易失性存储器，也叫 Flash EEPROM[1]，最初于 1984 年由东芝公司的舛岡富士雄（Fujio Masuoka）博士发明。早期由于价格昂贵，闪存最先被应用于嵌入式系统作为标准存储器。随着器件制作密度的提高和价格的降低，闪存被应用到笔记本电脑中代替磁盘，以及被应用到企业级存储的高端存储阵列中。目前随着大规模数据中心的电力和相关的冷却开销成为制约数据中心密度和能力增长越来越重要的因素，低能耗、高性能的闪存已被引入到以数据处理为中心（Data-Intensive Computing）的高性能计算系统中[3][6]。可以预见，闪存的应用将越来越广泛。

表1. DRAM、NAND 与 HDD 的比较[1]

	DRAM	NAND	HDD
读/写延迟	<0.1μs	~100μs	3000~5000μs
启动耗能（W/GB）	0.4	~0.01-0.04	~0.04
待机耗能	~25%启动耗能	<10%启动耗能	~20%启动耗能
擦除次数	无限制	10^4	无限制
数据保留时间	64ms	~10 年	>10 年
价格/容量	10 美元/GB	2 美元/GB	0.1 美元/GB

与内存所用的 DRAM[2]器件及磁盘相比，闪存在访问延迟、传输带宽、密度、价格和能耗方面弥补了 DRAM 和磁盘之间的差异。表 1 给出了三者在能耗、读写延迟、价格/容量比、可靠性方面的对比。可以看出，正是由于在价格、性能、能耗等方面的综合优势，才使得闪存的应用范围越来越广泛。

但是闪存也有结构上的缺点：每个存储单元只有在擦除以后才能写数据，并且擦除操作所需的时间比写操作多一个数量级；另外每个存储单元的擦除次数有限，而且随着密度增高，可擦除次数减少。这些缺点在应用闪存时不可避免，因此如何运用好闪存成为研究闪存的热点之一。

因为闪存的访问延迟、传输带宽、价格在 DRAM 和磁盘之间，而且密度比 DRAM 和磁盘高，能耗比它们低，所以闪存存储器在存储系统结构中应用于哪个层次也是研究热点之一。闪存在存储系统可应用的层次有三类：内存层、缓存层和外存储层。

- 内存层：将闪存用于内存层的研究方向主要有两类：一是将闪存和 DRAM 组成同一级别的混合内存[2]；二是将闪存明确地作为缓慢的二级内存分区供应用使用[3][47]。由于闪存在读写延迟上与 DRAM 相差较大，后一种方式的整体性能较高。而第一种方式的研究方向逐渐转到由读写延迟更接近 DRAM 的 Storage-class

1 Electrically-Erasable Programmable Read-Only Memory，电可擦可编程只读存储器
2 Dynamic Random Access Memory 动态随机存取存储器

memory(SCM)[4]与 DRAM 组成同一级别的混合内存[5]。

- 缓存层：由于缓存层处在内存层和外存储层之间，使得弥补 DRAM 和磁盘延迟差距的闪存特别适合作为缓存[18][51、52]。
- 外存储层：从最早将闪存作为嵌入式系统的外部存储器到现在将闪存作为高性能计算系统节点的存储设备[6]，将闪存做到外存储层的尝试一直是学术界和工业界研究闪存的重点。因为闪存与磁盘相比，具有高性能、低能耗、抗震等特点，将闪存作为外存储层可以提高系统的存储性能、访问速度、可靠性，降低系统能耗。目前将闪存作为外存储层所采用的软硬件有两种体系结构[56]：一是设计针对闪存的文件系统和闪存驱动，底层对应的是 Raw 闪存（裸闪存[3]）；二是在设备内设计闪存转换层（Flash Translation Layer, FTL），对外提供块设备访问接口，例如固态硬盘（SSD）。

将闪存作为外存储层是本文的关注点。本文首先简单介绍目前广泛使用的两类闪存，然后分别分析总结了基于闪存的外存储的两种体系结构各自适应的应用场景、优缺点以及需要研究的问题。最后对基于闪存的外存储技术进行总结展望。

2 闪存简介

闪存分为 NOR 闪存和 NAND 闪存两类，前者是将内部单元连接成“或非”门，后者是将内部单元连接成“与非”门。

二者的共同点是都将存储单元组织为块阵列。块是擦除操作的最小单位，擦除操作将块内所有的数位重置为“1”，对闪存单元的数据写入需要完成擦除操作之后才能进行。二者的异同点如下。

NOR 闪存以并行的方式连接存储单元，具有分离的控制线、地址线和数据线，具有较快的读速度，能够提供片上执行的功能，可以以字节为单位进行数据访问。但写操作和擦除操作的时间较长，且容量低、价格高。因此 NOR 闪存多被用于手机、BIOS 芯片以及在嵌入式系统中进行代码存储。

表2. NOR 闪存与 NAND 闪存特性对比

	NOR[54]	**NAND[55]**
读写单位	字节	页
读延迟	0.11μs	25μs
写延迟	11μs/word	250μs/page[4]
擦除延迟	600ms	2ms
存储密度（容量）	低	高
支持片上执行	支持	不支持
擦除次数	10^5	10^5

NAND 闪存以串行的方式连接存储单元，复用端口分时传输控制、地址和数据信号，并由一个复杂的读写（I/O）控制器为主机提供接口。NAND 闪存只能以页为单位进行数据访问。由于对一个存储单元的访问需要多次地址信号的传输，且每次访问 512B、2KB 或 4KB 的数据，NAND 闪存的读取速度较慢．但写操作和擦除操作比 NOR 闪存快，且容量大、价格较低，因此基于闪存的外存储器一般都以 NAND 闪存为主。

表 2 显示了 NOR 闪存与 NAND 闪存特性对比。

当前主流的 NAND 闪存分为 SLC（Single-level cell，单电平单元）和 MLC（Multilevel cell, 多电平单元）两种类型。前者每个存储单元（memory cell）只存储单一的二进制位（即

[3] 即闪存芯片直接连接到 CPU(SoC),并且能够直接被操作系统读取
[4] 此处涉及的 NAND 产品[55]一个 page 为 2KB，NOR 产品[54]一个 word 为 2B，因此一个 page 等于 1K word。

可表达 2 个值），二进制值由两个电压阈值来区分；后者每个存储单元可存储 2 个或 3 个二进制位（即可表达 4 个或 8 个值）。因此，MLC 比 SLC 容量更大，更便宜，但它的性能也比 SLC 差，且寿命更短。二者特性的对比如表 3 所示。

表3. SLC 和 MLC 芯片的区别[53]

种类	SLC NAND 闪存	MLC NAND 闪存
信息密度	一个单元只存储一个比特（低）	一个单元能存储两个比特（高）
使用寿命	长	短
读延迟	25μs	60μs
写延迟	250μs	900μs
擦除延迟	3.5ms	3.5ms
成本	高	低

3　闪存文件系统技术

在 Raw 闪存上开发闪存驱动和闪存文件系统的软件体系结构可以充分利用闪存特性进行设计，并利用文件系统信息为闪存存储提供更多提高性能、可靠性以及降低能耗的机会。缺点是需要对软件层和硬件层同时开发，增加了开发的代价。这种软件体系结构适应闪存作为嵌入式系统外存储器的场景，主要有三个原因：一是嵌入式系统体积小，能耗要求苛刻，对成本较敏感，因此宜采用 Raw 闪存而不宜采用有微控制器、闪存转换层的固态硬盘；二是传统文件系统基于磁盘特性设计，在 Raw 闪存上运行传统文件系统性能和可靠性差；三是在嵌入式系统上设计针对闪存文件系统的应用较在个人电脑、数据中心等场景下设计的周期短，代价低。

基于闪存的文件系统最早为微软在 1990 年左右设计的 MFFS[8]。其次较为成功的是瑞典 Axis 通信公司于 1999 年开发的一种专门针对 NOR 闪存存储器的日志文件系统 JFFS[9]。目前 JFFS 发展成为两个版本 JFFS1、JFFS2。其中 JFFS1 应用在 Linux2.2 以上版本中，JFFS2 应用在 Linux2.4 以上版本中。YAFSS1[10]是早期针对 NAND Flash 的文件系统中较为成功的，它由 Aleph One 公司于 2002 年开发，3.5 节将以其为例细述闪存文件系统技术。由 IBM 和诺基亚的工程师格莱克斯纳（Thomas Gleixner）、比邱茨基（Artem Bityutskiy）等人于 2006 年发起设计的 UBIFS[11]可以看作是 JFFS3，其开发着重于性能和扩展性的提高，是目前基于闪存文件系统中较为成功的。基于闪存的文件系统均运行在闪存驱动上层，因此本章先简单介绍目前两种主流的闪存驱动，其次分析总结基于闪存文件系统的关键技术和研究热点。

3.1 Flash 驱动

当前主流的闪存驱动有 MTD（Memory Technology Device）[12]和 UBI（Unsorted Block Images）[13]。MTD 的作用是对不同类型和型号的存储设备提供统一的读写管理接口，完成文件系统对闪存芯片的访问控制。UBI 实现在 MTD 上层，是针对闪存设备的卷管理系统，可以在一块物理闪存设备上管理多个逻辑卷。

表4. UBI 与 MTD 特性比较

特性/类型	UBI	MTD
支持读写擦除操作	是	是
包含擦除块	是	是
实现磨损均衡	是	否
支持处理数位翻转	是	否
对卷自动创建、删除、重定义大小	是	否
支持卷更新操作	是	否
支持自动改变逻辑擦除块的操作	是	否

UBIFS 就是基于 UBI 的文件系统。UBI 类似逻辑卷管理（LVM，Logical Volume Management），除了将逻辑擦除块映射到物理擦除块，还实现了全局的擦写平衡和运输读写错误处理。一个 UBI 卷是逻辑擦除块（Logic Erase Block, LEB）的集合，每一个逻辑擦除块都可以映射到任

意物理擦除块（Physical Erase Block, PEB），这些映射由 UBI 管理，对用户是透明的。UBI 的特点为：

1. 提供可以自动创建、删除和重定义大小的卷
2. 实现了在整个闪存设备的磨损均衡（wear-leveling）
3. 透明地处理坏掉的物理擦除块
4. 可以最大限度减少由于数位翻转（bit-flips）引起的数据丢失

UBI 与 MTD 特性的比较如表 4 所示。

3.2 文件系统数据存储方式

闪存擦除后才能进行写操作的特性使得针对闪存的存储都采用异地更新的策略。JFFS 文件系统采用日志结构（log-structure，如图 1（a）所示），在闪存上顺序存储元数据和数据。这种日志式存储消除了随机写，使得写性能提高，但是性能对垃圾回收算法依赖性高。一些文件系统（如 YAFFS、UBIFS 等）没有强调采用顺序存储方式，而是将元数据和数据分开存放，更新时采用“写时复制”（copy-on-write）（如图 1（b）所示）的形式：将更新的数据写入到新地址并建立与元数据的连接，同时取消过时数据与元数据的连接。这种存储方式也避免了原地更新（in-place update），有利于在闪存中建立索引，虽然不能消除随机写，但是读性能要优于日志式存储。另外，NAND 闪存的物理页除了包含数据区域，还包含用于存储控制信息的辅助区（spare area）。基于 NAND 闪存的文件系统（例如 YAFFS）利用辅助区存储该页的元数据，减少存储空间和加载时间。

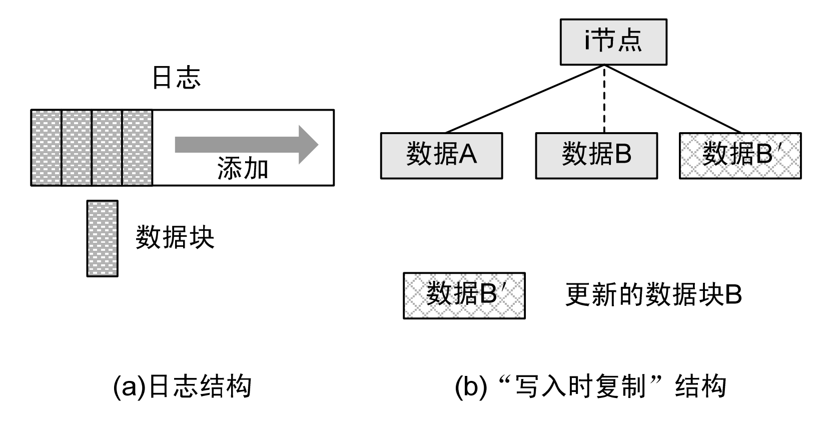


图1. 两种数据存储方式

综上，研究出适合闪存异地更新特性的数据存储方式是提高文件系统性能的关键之一。

3.3 文件系统索引的数据结构

文件系统在运行时需要在内存和闪存中建立索引，以便找到数据在闪存中的存储位置。早期的文件系统（MFFS、JFFS1）只在内存中建立索引，因此加载时需要将整个文件系统的索引加载到内存，这使得系统加载时间长、内存占用多，而可靠性低。内存中索引主要是线性的链表结构，一个文件内的数据块以链表的形式存储，查询节点时也非常耗时。随着闪存价格的降低，从 JFFS2 以后的闪存文件系统将索引存储在闪存中，内存中只保存部分索引，降低了加载时间和内存占用。在 JFFS2 中，索引结构也由链表改成了哈希表，减少了查询时间。从 YAFFS1 开始，内存的索引结构改成查询时间更短的树结构。而在 UBIFS 中，不仅内存中的索引结构为树结构，在闪存上也采用 B+树建立索引结构。在该 B+树中只有叶子节点存储文件系统的数据，其余节点均属于索引节点。由于采用异地更新策略，因此对 B+树中的任何叶子节点的更新都导致与该节点相关的直到根节点的整个“树枝”的更新，所以又将这种针对闪存使用的 B+树称为“漫游树（Wandering tree）”。图 2 显示了在漫游树上更新的示例。该树共有 6 个节点：A-F。如果更新节点 F，则需要在一个新页中写入更新数据 F′，但是 C 节点指向 F 节点，因此 C 节点需要更新为 C′ 节点以指向 F′ 节点。基于此，

该更新过程需要一直更新到 A′ 节点才能结束。这种更新开销是非常大的，为此，UBIFS 采用日志的形式记录对索引节点的更新，先将索引更新写到日志(journal)中，隔一段时间后提交，减少了在闪存上的索引的频繁更新。但是这种一个节点对应闪存芯片一个物理页的更新代价还是很大，因此有人提出了μ树[14]。μ树是一个扩展的 B+树，它的逻辑结构与 B+树相同。但是物理结构中，B+树每个节点占闪存的一页而μ树将一个路径中从根节点到叶子节点均存储在闪存的一页中。因此在μ树中更新数据只需对一个页进行操作，降低了开销。图 3 显示了在μ树上更新过程示例。

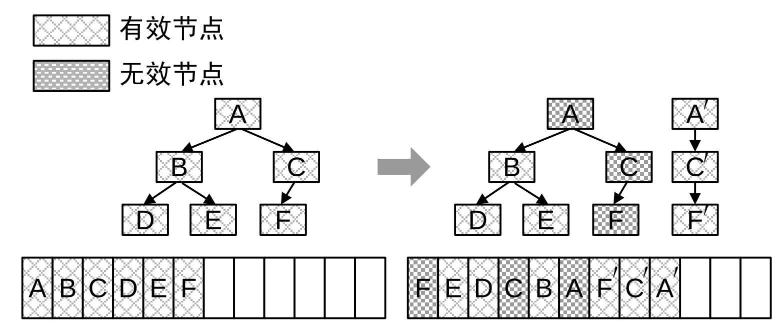


图2. 在漫游树上更新过程示例[14]

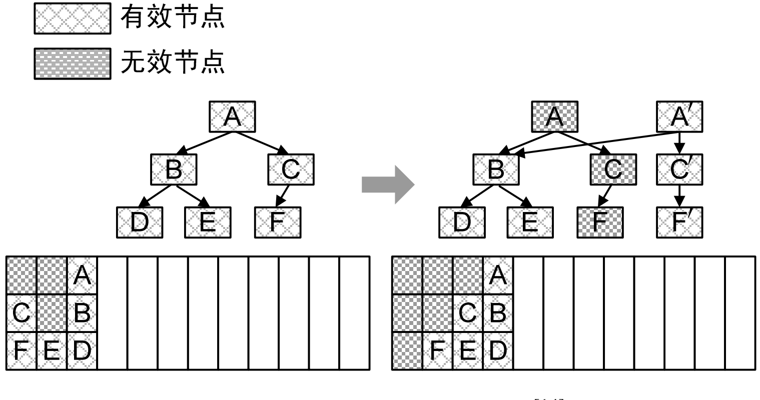


图3. 在μ树上更新过程示例[14]

综上，文件系统索引的数据结构主要以减少对节点查询、更新等操作的开销为原则，目前的研究主要集中在对 B+树的设计和实现上的优化。但是 B+树是基于磁盘而设计的，因此对于基于闪存特性重新设计新的索引数据结构，也会成为日后研究的方向。

3.4 利用文件系统层信息实现垃圾回收和磨损均衡

由于基于闪存的存储如果都采用异地更新的策略会导致系统运行过程中无效数据的增长，因此必须考虑垃圾回收的机制以使得系统能持续运行。闪存的存储单元都有擦除次数的限制，并且随着闪存密度的增加，闪存的最大擦除次数会减小。比如一个 SLC 的擦除次数最高可为 10^5，MLC 的擦除次数则最高为 10^4[15]。因此，无论哪种基于闪存存储的软件体系结构都必须考虑垃圾回收和磨损均衡算法。本节侧重分析利用文件系统层次信息实现垃圾回收和磨损均衡，4.2 和 4.3 节会分别对这两种机制的算法本身进行分析总结。

文件系统层拥有存储设备层没有的语义信息，包括应用层的读写操作信息以及文件的组织方式等，因此利用这些信息可以使垃圾回收或磨损均衡算法的实现更加有效。如何利用这些信息设计垃圾回收和磨损均衡算法是基于闪存的文件系统获得更高性能的关键。例如 CFFS[16]利用文件系统层信息将闪存的擦除块分成元数据块、数据块、空闲块三类。因为元数据更新比数据频繁，元数据无效的频率也会比数据频繁。将元数据单独存放可以在垃圾回收时所需复制的数据量最少（仅复制元数据），减少了开销。但这导致元数据块比数据块的擦除次数更多，因此，CFFS 记录每次某个块存储的数据类型，如果此次是元数据块则擦除后将其设置为数据块，下次就用于存储数据，这样就可实现磨损均衡。SFS[17]利用文件系统层信息将文件数据块依照被再次更新的可能性（定义其为“热度”）的大小分组，在垃圾回收时依据热度选择回收块，使更新次数多的块（含有效页少）容易回收，达到垃圾回收的开销最小。

3.5 案例

本节通过描述一个典型的闪存文件系统 YAFFS1 的实现，使读者对上述的闪存文件系统技术有更好的了解。

YAFSS1 是 2002 年由 Aleph One 公司开发的专门针对 NAND 闪存的文件系统，可以用在 Linux 和 Windows CE 平台上。在 Linux 中，YAFFS1 提供直接访问文件系统的应用程序接口（API），可以不依靠 MTD 和虚拟文件系统（Virtual File System，VFS）（当然，也可以利用 MTD 和虚拟文件系统）。图 4 显示了 YAFSS1 在文件系统中的位置。YAFFS1 采用了针对 NAND 闪存特点的数据管理方式，以页为单位存储数据。它支持一个页包含 512 字节的数据和 16 字节的辅助区的 NAND 闪存。在 YAFFS1 中页被称作 chunk（段），32 个 chunk 构成一个 block（块）。利用 chunk 中的辅助区中的 8 个字节来存储 chunk 的归属信息，利用的信息如表 5 所示。

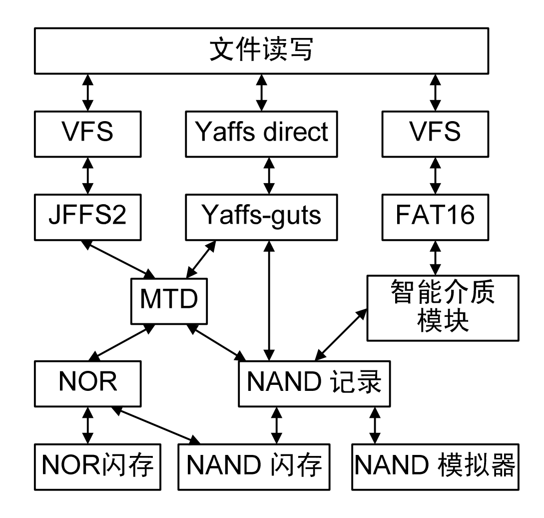


图4. YAFFS1 在文件系统中的位置[7]

YAFFS1 在闪存上存储每个文件的 inode，一个文件的 inode 占用闪存中一个完整的页，也就是一个 chunk。在文件系统加载时只须扫描每个页的辅助区，就可以在内存中建立文件系统镜像，因此加载时间较快。YAFFS1 在内存中建立的是层次结构的索引，称为文件树，每一个文件对应一个文件树。该文件树是由 Tnode 结点组成的。一个 Tnode 为 32 B 内存区，文件树的中间结点为 8 个 4 B 的指针指向比它更低一层结点的内存地址；叶结点为 16 个 2 B 的指针存储页物理地址。文件树的高最多为 6 层，这主要是限制对树操作的开销，但也因此限制了存储的文件大小。在极端情况下文件树为一个完全的 8 叉树，叶子结点共 86 个，假设页大小为 N 字节，则文件大小最多为 $N\times2^{18}$ 字节，可满足大多数应用。运行时 YAFFS1 维护文件树，因而实现了占用内存少，查询速度快，读写速度快。

表5. YAFFS1 对闪存页的辅助区利用[19]

位数	内　容
20	ChunkID，该页在一个文件内的索引号，所以文件大小被限制在 2^{20} 页，即 512Mb
2	2 位的序列号
10	ByteCount，该页内的有效字节数
18	ObjectID，文件 ID 号，用来唯一标示一个文件
12	Ecc，Yaffs_Tags 本身的 ECC 检验和
2	Unused，保持为 1

YAFFS1 在每次写入新数据时判断是否需要进行垃圾回收，当系统的空闲块小于某一阈值时启动垃圾回收，否则不进行。YAFFS1 的垃圾回收策略采用将贪心策略和随机选择策略按一定比例混合使用：当满足特定的小概率条件时，垃圾回收会试图随机选择一个可回收的页面；其他情况下，使用贪心策略回收最“脏”的块。

3.6 小结

本节概述了目前的闪存文件系统中研究热点各自采用的关键技术，下面通过表 6 对本节内容进行总结。

表6. 闪存文件系统关键技术总结

研究热点		关键技术	关键点
存储方式		日志结构	采用日志、异地更新；能消除随机写；读性能差
		写时复制	不采用日志、异地更新；不能消除随机写；读性能优
索引结构	内存中	链表	结构简单，查询时间长
		哈希表	查询时间短，需要解决键值冲突
		树	查询时间适中，利于索引大规模数据
	闪存上	漫游树	利于减少系统加载时间，数据更新时需更新整个“树枝”所在页
		μ 树	利于减少系统加载时间，数据更新时只需更新一个页
利用文件系统层信息		CFFS	元数据与数据分开存储在不同物理块中，利于垃圾回收和磨损均衡
		SFS	文件数据块依照被再次更新的可能性的大小分组。利于垃圾回收

4　基于闪存的固态硬盘技术

固态硬盘通过内部的闪存转换层，对外提供块设备访问接口，对已有的文件系统和应用程序完全兼容，减少了开发和测试新软件的代价，是闪存技术能迅速得到广泛应用的重要因素之一。但将其加入到传统基于磁盘设计的文件系统和应用之下，不能充分利用闪存技术的特点使整体性能最优。由于与传统存储体系结构兼容，目前固态硬盘的应用集中在个人电脑、数据中心等场景中，用以全部或部分代替磁盘作为系统的外存储设备。闪存转换层作为固态硬盘的关键技术，自然成为研究重点。本节分析和总结闪存转换层的关键技术，并介绍了一种基于固态硬盘的新型体系结构。

4.1 地址映射

地址映射机制是用来建立逻辑地址和闪存存储器的物理地址之间的映射关系，是将上层块请求转换成固态硬盘内部请求的关键。为了提供可接受的写性能，闪存转换层采用异地更新的机制，将擦除和写操作分离，有效提高了写性能，但要求动态的地址映射机制的支持。根据映射表的粒度不同，可分为基于页的地址映射、基于块的地址映射、混合式地址映射和变长式地址映射。

4.1.1 基于页的地址映射

第一个闪存转换层机制由班（Ban）等人 1995 年设计[20]，并在几年后被 PCMCIA 采用作为基于 NOR 闪存的闪存转换层标准[21]。它采用的是基于页的映射机制：地址映射表以页为粒度，包含的表项数与闪存存储器的页数相同，图 5(a) 显示了这种结构的映射图。为了减少内存的占用，该闪存转换层将整个映射表存储在闪存中。这种机制对于以字节为寻址单位的 NOR 型闪存是适用的。通过在必要的时候分配替代页表来更新映射页，映射页可以更新。DFTL[22]是第一个将上述的基于 NOR 型闪存的闪存转换层机制转换到 NAND 型闪存的闪存转换层，省略了替代页部分。但是因为将所有的改变存储在内存中，一旦系统宕机需要扫描整个闪存设备中的数据信息以使系统达到一个一致性的状态。因此 DFTL 的可靠性低。LazyFTL[23]在基于页映射的基础上改善了可靠性，并减少了内存占用。它将闪存分为数据块区域（DBA）、映射块区域（MBA）、冷数据区域（CBA）、更新块区域（UBA），其中数据块区域、更新块区域相当于在闪存中设置的写缓存。这样就可以在内存（RAM）中仅维护少量的更新映射表和部分全局映射表，实现基于页的映射机制。LazyFTL 避免了大量内存的占用，却能保持映射的灵活和性能。基于页的映射粒度在所有的映射粒度中灵活性最高、性能最好，但所需的内存空间较大。尤其随着闪存存储器的容量不断增大，基于页的地址映射

对内存大小的要求难以得到满足，为此有了下面几种方式的诞生。

4.1.2 基于块的地址映射

在基于块的地址映射机制中，逻辑地址包含逻辑块号和块内偏移两部分。地址映射表只需要保存逻辑块到物理块的映射，逻辑块和物理块的块内偏移相同。图 5(b) 显示了这种结构的映射图。NFTLs[25]是一类典型的基于块的地址映射机制。它们为每个需要更新的数据块维护一个或多个替换块，用来存储更新的数据。由于块内顺序存储的限制，对某页的多次更新将占用多个替换块，导致日志块的空间利用率很低。可见，块级映射的优点是地址映射表规模小，节省存储空间；缺点是因粒度太大而缺乏灵活性，导致闪存的空间利用率低。

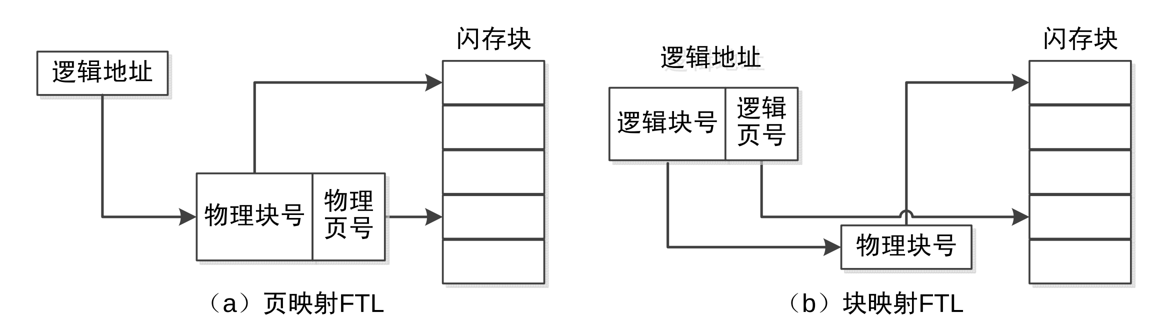


图5. 闪存转换层的地址映射机制[24]

4.1.3 混合式地址映射

混合式地址映射机制指的是对频繁更新的数据维护基于页的地址映射表，而对大量的其他数据维护基于块的地址映射表。这种映射机制利用较少的内存空间开销提供了高灵活性和高性能，因此是目前应用最广泛的映射机制。这类映射机制的早期思想是将闪存分成数据块和替换块，替换块作为更新的缓存维护基于页的映射而数据块维护基于块的映射。BAST[28]是第一个采用混合式地址映射机制的闪存转换层。它克服了 NFTLs 中对替换块顺序存储的限制，更新的数据按写操作的先后在对应的替换块中依次存储，提高了替换块的空间利用率。由于在每页的空闲区中存储对应的逻辑地址，内存缓存中地址映射表的大小与基于块的地址映射机制相同。但是由于替换块与数据块一一对应，导致 BAST 策略很容易因为频繁更新的数据块用光空闲的替换块而引发垃圾回收，这被称为块的“thrashing(颠簸)”问题。为了解决这个问题，FAST[29]机制被提出。FAST 进一步解除对替换块的限制，允许任何数据块的更新数据存储在任意替换块的任意位置。这进一步提高替换块的空间利用率，降低垃圾回收的频率，提高了性能。

因为数据在替换块中是乱序存储，在数据块中是顺序存储，因此混合式地址映射在将替换块和数据块中的数据合并时会产生三种合并操作：交换合并、部分合并、全合并。当替换块中的数据全部更新，对应的数据块中数据全部失效时，只需将替换块置成新的数据块，作废原先的数据块，这种合并为交换合并。当替换块中的数据部分顺序更新、对应的数据块依然有有效数据时，需要将数据块中的有效数据写入到替换块中，再将替换块置成新的数据块，作废原先的数据块，这种合并为部分合并。当替换块中的数据部分无序更新，对应的数据块依然有有效数据时，需要分配新的空闲块，将替换块和数据块中有效数据顺序写入到新的空闲块，原先的替换块和数据块作废，这种合并操作为全合并。从合并过程中可以看出部分合并和全合并是混合式地址映射独有的，并且全合并的开销最大，要尽量避免。但是，随着数据在替换块中的顺序被打乱，出现全合并的概率就会加大。因此后来出现的混合式地址映射的闪存转换层以平衡替换块空间利用率和减少合并操作的开销为目的。超级块策略[30]和

SAST[31]策略均是将 N 个数据块对应 K 个替换块。不同的是超级块策略在物理块的空闲区保存基于页的映射表，SAST 策略限制日志块的数量并将基于页的映射表保存在内存中。这两种策略的共同问题是需要提前根据访问模式调节参数（N、K 等），因为访问模式改变可能会导致它们的性能变差。LAST[32]策略将替换块分成不同的功能区，以充分利用替换块空间，同时保证垃圾回收开销尽可能小。例如对于大的写请求，数据写到顺序日志区中，对于频繁更新的“热”数据写到热分区的随机日志中，其他的写请求被写到“冷”分区中。

HFTL[33]策略改变了以前的混合式地址映射机制的基本思想，它没有将基于页映射的块作为更新的缓存，而是采用基于哈希的热数据鉴别技术[34]，使得被鉴别为存储频繁更新的“热数据块”采用基于页的地址映射机制，其余块采用基于块的地址映射。这种策略达到了真正频繁更新的数据块采用基于页的地址映射的混合式地址映射机制的目标，但是随着访问模式的改变，一些热数据块可能不再频繁更新，需要从基于页的映射变为基于块的映射，这会增加额外的开销。

4.1.4 变长式地址映射

基于μ树结构的μ-FTL[35]可以依据写请求的大小调节映射的粒度。μ-FTL 利用μ树的性质对于大的顺序写请求的映射采用粗粒度，而对于小的随机写请求的映射采用细粒度，使得整体的映射机制既保持了灵活性又减少了内存占用。严格地说，这种变长式地址映射也是混合式地址映射的一种，但是不同于混合地址映射人为地固定映射粒度，μ-FTL 可以根据每个写请求的特征为其分配适合的映射粒度，对负载更具适应性。然而这种变长式地址映射只能依靠树的形式实现，因此地址映射会增加这类映射机制的额外开销。

设计高效的地址映射机制是保证固态硬盘性能的关键。从本节所述可以看出闪存转换层地址映射主要以既保证映射的灵活性、有效性，又减少内存的占用为主要的设计原则和研究方向。

4.2 垃圾回收

根据 3.4 节所述，基于闪存的存储系统由于采用异地更新，因此都需要考虑垃圾回收机制。垃圾回收过程首先选择一个块作为回收对象，将其中的有效页复制到一个空闲块中，然后更新地址映射表，擦除该块，最后把它加入空闲块列表中。由于垃圾回收的过程涉及数据复制、擦除等操作，垃圾回收机制的优劣影响固态硬盘的读写性能和可靠性，因此成为固态硬盘性能瓶颈。在目前基于闪存的存储技术（包括闪存文件系统和闪存转换层）的基础上，从垃圾回收的过程可以看出：垃圾回收算法中最需要考虑的关键问题是回收块选择。

4.2.1 回收块的选择：

选择什么特征的块进行回收不仅影响垃圾回收的收益，还影响对闪存的磨损均衡。因此回收块的选择是基于闪存存储技术研究的重点问题。回收块选择算法应以增大一次垃圾回收的收益以便尽可能减少对随机写操作的影响，并且尽可能使块磨损均衡以增加闪存的寿命为原则。目前可总结为两类思想。

第一种思想为贪心算法，每次选择有效页数最少的块进行回收。这种算法可使一次垃圾回收回收到最多的空间，使得收益最大，但不一定能保证磨损均衡。贪心算法实现简单，因此很多注重地址映射机制的闪存转换层选择该算法，至于减少垃圾回收的开销和磨损均衡可通过设计有效的地址映射机制间接实现。例如，μ-FTL 直接选择有效页数最少的块进行回收。LazyFTL 定义了块的开销（cost）公式（见公式 1），选择 cost 最低的块进行回收。

$$Cost = \left(C_{\text{read}} + C_{\text{write}}\right) \times N_B + C_{\text{erase}} \quad \text{（公式 1）}$$

式中：C_{read}、C_{write}、C_{erase} 是读、写、擦除延迟，N_B 是该块中有效页的数量

从公式 1 中可以看出 LazyFTL 的回收块选择算法依然属于贪心算法。超级块策略的回收块选择算法整体思想为贪心算法，但是考虑了垃圾回收过程中避免全合并的操作。它优先从没有有效页的块中选择能进行交换合并的块进行回收，如果没有则从最近最少更新的块中选择能进行部分合并的块进行回收，如果依然没有则选择有效页最少的块进行全合并回收。

第二种思想可以总结为效益-开销比算法，选择该值最大的块进行回收。依据公式定义不同，不同的算法选择的回收块侧重点不同。在川口（Kawaguchi）等人设计的一个基于闪存的转换层[36]中，收益/开销比（***Benefit/Cost***）的定义如公式 2 所示。

$$Benifit / Cost = age \times \left(1 - u\right) / (2u) \quad \text{（公式 2）}$$

式中：age 是该块最后一次改变的时间，u 是有效页数的比例

该公式可以保证回收到开销小收益大而且更新频率低的块，这种算法可以间接降低垃圾回收的次数，实现磨损均衡。

威尔斯（Wells）等人设计的一个基于磨损均衡的擦除算法专利[37]利用公式 3 选择回收的块。

$$score = W_1 \times obsolete + W_2 \times \left(\mathbf{max}\left\{erasures\right\} - erasures\right) \quad \text{（公式 3）}$$

式中：$obsolete$ 是该块无效页的数量，$\mathbf{max}\left\{erasures\right\}$ 是当前所有闪存块中最大的擦除次数，$erasures$ 是该块已经擦除的次数，W_1、W_2 是权重，二者之和应为 1

权重值可以依据空闲块的多少预先设定。当空闲块少时 $\boldsymbol{W_1}$ 值较大，侧重垃圾回收的效益；当空闲块多时 $\boldsymbol{W_2}$ 值较大，侧重垃圾回收的磨损均衡。这种算法在回收块选择时既考虑了效益又考虑了磨损均衡，但缺点是权重值需要预先设定，不能适应系统的变换。为此算法 CICL（cleaning integrated with cycle leveling）[38]被提出。CICL 算法在公式 3 的基础上将权重改为依据当前闪存块中擦除次数的差值计算出来的单调函数，从而可以自适应系统的变化。

还有一种回收块选择策略[39]采用了上述两种思想。当系统的空闲块下降到一定阈值时采用贪心算法回收块，否则采用第二种思想。

从本节中可知，回收块的选择需要平衡回收的效益和磨损均衡，既减少垃圾回收的开销又延长基于闪存存储系统的使用寿命。

4.3 磨损均衡

依据 3.4 节所述，为了实现较长的寿命和良好的可靠性，磨损均衡算法是基于闪存存储技术必须考虑的问题，自然也是闪存转换层技术研究的重点。磨损均衡算法的目标是尽可能平均每个闪存擦除块的擦除次数以增加闪存的使用寿命。从 4.2 节中可知，由于垃圾回收过程中涉及到擦除操作，磨损均衡是垃圾回收算法需要考虑的重要方面。但如果仅在垃圾回收过程中考虑，无法在闪存芯片上所有的块范围内实现磨损均衡。因此还需要设计全局的磨损均衡算法，本节总结了这部分技术（包括闪存文件系统和闪存转换层）。

目前全局的磨损均衡算法的主要思想是每隔一定时间，将闪存存储器上的冷、热数据进行交换以达到磨损均衡的目的。早期采用随机选择目标的形式，每隔一定数目的擦除操作随机地选择一个块进行擦除。JFFS 系统即采用这种方式，每执行 100 次擦除，系统随机地回收一个只包含有效页面的块。这种随机选择擦除块回收的算法实现简单，不需要记录每个擦除块的擦除次数，但是磨损均衡的实现是粗粒度的，没有针对性。洛夫格林（Lofgren）等人在其设计的一个磨损均衡的专利[40]中，基于擦除次数实现静态磨损均衡。该方法保留一个空闲块，当系统当前块中擦除次数最大值与最小值之差超过一定阈值时，将擦除次数最少的块的数据复制到空闲块中，擦除次数最大的块的数据复制到擦除次数最少的块中，回收擦除次数最大的块作为空闲块。这种方式实现的磨损均衡更有针对性，但是需要记录每个擦除块的擦除次数信息。为了减少内存空间的占用，一种基于块组的静态磨损均衡算法（group-based wear leveling algorithm）[41]被提出。它将 n 个擦除块看成一个组，在内存中只保留块组的擦除次数。这种方式需要选择合适的块组大小，以及有效地利用块组的擦除次数信息实现更有针对性的磨损均衡算法。

4.4 案例

本节通过一个典型的闪存转换层：LazyFTL 的具体实现，使读者对上述总结的基于闪存的固态硬盘技术有更好的了解。

LazyFTL 采用基于页的地址映射，但是通过采用一个更新的缓存（buffer，就像在混合式地址映射中的替换块）提供灵活高效的地址映射，同时又保持了可靠性和一致性。如 4.1.1 节所述 LazyFTL 将闪存存储阵列分为四部分：数据块区域、映射块区域、冷数据区域、更新块区域。除了映射块区域以外，其他部分都用来存储用户数据。图 6 显示了 LazyFTL 的结构。全局映射表（GMT）映射数据块区域与逻辑块的对应关系，全局映射表以页为粒度存储在映射块区域中。 FTL 中的 SRAM 采用最近最少使用算法(LRU，Least Recently Used)缓存一部分全局映射表以提高映射速度。第二级映射表——全局映射目录（GMD）存储在 SRAM 中，它维护全局映射表中所有有效页的映射信息。冷数据区域用来存储冷数据、更新块区域用来存储更新的数据。因此 LazyFTL 通过冷数据区域和更新块区域，延迟对存储在闪存上的全局映射表的改变。更新块区域中设有当前更新块（CUB）用于写操作。当前更新块写满后，另一个空闲块被分配变成新的更新块。同样，在冷数据区域中，有一个当前冷数据块（CCB）处理移动数据页操作。LazyFTL 为冷数据区域和更新块区域的数据建立另外一个基于页的地址映射表——更新映射表（UMT），更新映射表以哈希表或搜索树等方式存储在 SRAM 中。由于 LazyFTL 将冷数据区域和更新块区域的擦除块数量设置得很小，因此更新映射表的实体很小，对其操作的开销很低。

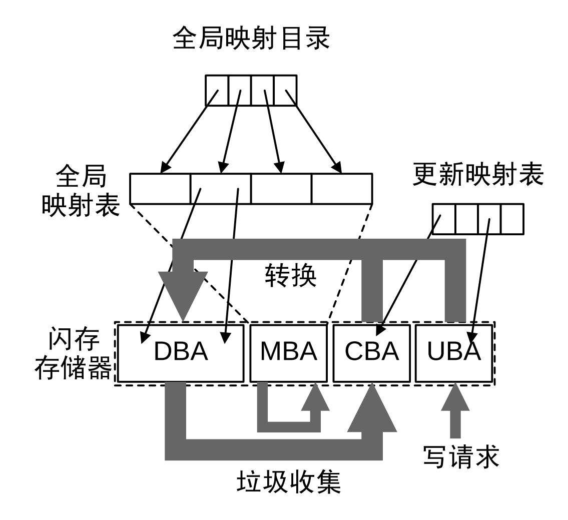


图6. LazyFTL 的结构

当冷数据区域或者更新块区域的擦除块已达到规定数量时，需要执行转换操作（Convert Operation），步骤如下：（1）在冷数据区域或者更新块区域中选中一个目标块；（2）将更新映射表中该块有效页对应的映射删除；（3）在全局映射表中建立新的映射；（4）直接将目标块转换成数据块区域中的数据块，因为在 LazyFTL 中数据块区域的数据块也是乱序存储的。

这种转换操作的开销仅体现在更改全局映射表中，因此这种操作的开销比混合映射机制中的全合并操作的开销小得多。转换操作中所选的块为需要更改全局映射表中表项最少的块，即采用贪心算法。LazyFTL 的垃圾回收操作针对数据块区域和全局映射表区。当系统中空闲块的数量小于一定阈值时，触发垃圾回收操作。垃圾回收块的选择采用贪心算法，具体实现在 4.2 节中已描述。值得关注的是当选中的回收块是数据块区域时，意味着其中的有效页相对于无效页更“冷”，因此将其中的有效页复制到冷数据区域区中的 CCB。

LazyFTL 没有实现针对性的磨损均衡算法，不过通过擦除块在上述四个区域的轮换，能一定程度上达到磨损均衡。

通过上面对 LazyFTL 实现的描述可知：闪存转换层中地址映射机制、垃圾回收机制和磨损均衡机制不是孤立的而是相互联系、相互约束的。因此在设计闪存转换层时，应全面考虑，使得闪存转换层整体性能最优。

4.5 ioMemory[43]：一种新型体系结构

相当长时间以来，固态硬盘总是将闪存转换层做在设备内部，使设备的上层软件可以像使用磁盘一样使用固态硬盘。但是这也导致了数据流需要经过磁盘阵列（RAID）控制器、嵌入式处理器等层进行不必要的数据转换和队列等待才能进入闪存，这必然增加访问的延迟。但是固态硬盘经过长时间的开发应用，已经有相当成熟实用的技术。为了既能保留固态硬盘本身已有的技术优势又能减少数据流程，Fusion-IO 公司 开发了 ioMemory 平台。ioMemory 平台是基于固态硬盘的一种新型体系结构，其与传统固态硬盘架构的对比如图 7 所示。从图中可以看出 ioMemory 架构是将针对闪存特征采取的算法（Remapping、Wear-leveling 等）从固态硬盘的内部闪存转换层移到主机的软件层—虚拟闪存存储层（VSL，Virtualized Flash Storage Layer），使得数据不必在磁盘阵列控制器、嵌入式处理器等层中进行不必要的数据转换、队列等待等，减少数据处理流程，降低延迟和能耗。图 8 显示了传统固态硬盘和 ioMemory 方式数据流程对比，从图中的数字可以看出 ioMemory 方式

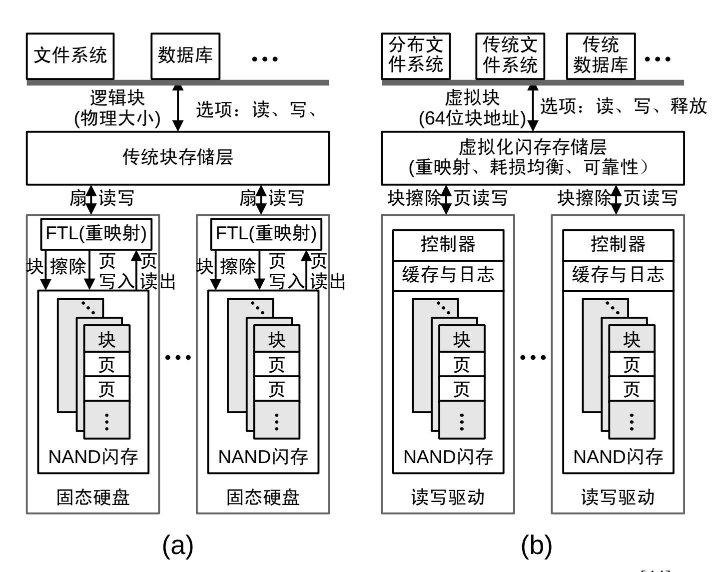
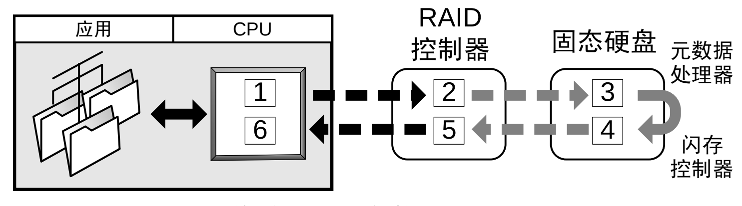
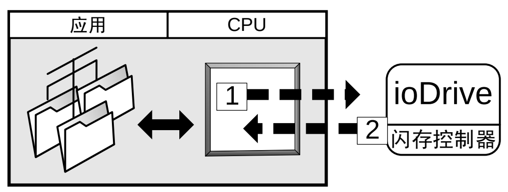


图7. 传统固态硬盘架构(a)和 ioMeomory 架构(b)[44]



(a) 传统固态硬盘数据流程



(b) ioMemory数据流程

图8. 传统固态硬盘和 ioMemory 方式数据流程对比[45]

的数据处理流程少于传统固态硬盘。虚拟闪存存储层在 ioMemory 平台处于系统的内核中，实现了以下功能：执行对闪存的重映射、磨损均衡等算法；用日志结构分配策略隐藏擦除块的延迟；处理坏页恢复；为上层提供虚拟的 64 位块寻址；保持对现有的文件系统和数据库等应用系统的兼容。虚拟闪存存储层可以看成将闪存转换层从固态硬盘移到主机的软件层。这样做的好处是：可以为每个 CPU 核提供对闪存的独立并行的直接访问，减少数据流程，降低延迟；容易实现针对硬件的新特性和性能更新；依靠主机 CPU 和内存处理算法，容易通过主机 CPU 和内存的更新而提升性能；简化了存储流程，减少了引起宕机的地方，提高了可靠性。

目前基于 ioMemory 平台设计的文件系统 DFS[44]（Direct File System）处于虚拟闪存存储层的上层，利用虚拟闪存存储层的功能实现对闪存硬件设备的存储管理。因此 DFS 相比其他文件系统实现简单，性能高。ioMemory 平台是对基于闪存的新的体系结构的一次成功的尝试，但是目前它的技术被公司垄断，价格昂贵。

4.6 小结

本节通过表 7 对第 4 节所提到关键技术进行总结。

表7. 基于闪存的固态硬盘技术

<table>
<tr><th colspan="2">研究热点</th><th>关键技术</th><th>关键点</th></tr>
<tr><td colspan="2" rowspan="4">地址映射</td><td>基于页的地址映射</td><td>地址映射以页为粒度,灵活但占用内存</td></tr>
<tr><td>基于块的地址映射</td><td>地址映射以块为粒度,占用内存最少但不灵活，闪存空间利用率低</td></tr>
<tr><td>混合式地址映射</td><td>频繁更新的数据维护基于页的地址映射表，其余数据维护基于块的地址映射；灵活性、内存占用和闪存空间利用率都适中，但是全合并操作不能避免</td></tr>
<tr><td>变长式地址映射</td><td>根据每个写请求的特征为其分配合适的映射粒度，大的顺序写请求的映射采用粗粒度，小的随机写请求的映射采用细粒度</td></tr>
<tr><td rowspan="2">垃圾回收</td><td rowspan="2">回收块选择</td><td>贪心算法</td><td>选择有效页数最少的块进行回收，可使一次垃圾回收收益最大，但没有考虑开销和磨损均衡</td></tr>
<tr><td>效益-开销比算法</td><td>综合考虑垃圾回收收益、开销和磨损均衡选择回收块，不同算法对这三个因素的侧重不同</td></tr>
<tr><td colspan="2" rowspan="2">磨损均衡</td><td>随机磨损均衡</td><td>每隔一定时间，随机选择块擦除，没有针对性</td></tr>
<tr><td>基于擦除次数的磨损均衡</td><td>每隔一定时间，将擦除次数最多和最少的两块中的数据互换</td></tr>
</table>

5　总结展望

本文按照目前管理闪存的两种软件体系结构分类分析，较为全面地总结了当前基于闪存的外存储技术。文中指出，由于闪存本身的特性，无论采用哪种软件体系结构，异地更新策略、垃圾回收和磨损均衡算法都是使用闪存不可避免的问题。因此，设计更加有效的算法依然会成为日后研究闪存的热点。

虽然当前基于闪存的外存储技术已经相对成熟，但是依然存在缺陷，若想充分发挥闪存存储器的最大价值，未来需要在以下两个方面取得突破：

一是设计闪存存储器新的体系结构。目前管理闪存的这两种软件体系结构都不能既充分利用闪存特性又保证开发代价最小。因此，设计基于闪存的完整的硬件软件体系结构以充分利用闪存特性并保证对应用的尽量兼容，达到最优的性价比应成为日后突破的方向。

二是对闪存外存储器所在层次的重新定位。CPU 的性能每年提高约 60%，而内存子系统每年只增长 10%，磁盘增长的则更少。磁盘每 24 个月就会在容量上翻番，但在过去 15 年中，其性能只提高了 10 倍[46]。这导致处理器与存储系统之间的性能差距 ，而且差距越来越大。为了解决这种性能差距，长期以来都是在存储系统上进行分层存储，每层采用不同的存储介质，顶层的体积小、速度快，与处理器最接近；而较低的存储器层，虽然容量在不断增加，但运行速度要慢得多，同时还要消耗更多的电力。这种做法虽然一定程度上缓解了处理器和存储系统的性能差距，但是大量的能源被大型存储器配置消耗。闪存相对磁盘在性能上有了极大的提高，因此可以打破传统存储系统的分层结构，将 CPU、DRAM、闪存存储器重新定位，使得整体计算机系统高效、节能。

参考文献：

[1] Emerging Research Devices Reports, International technology roadmap for semiconductors, 2011 Edition, http://www.itrs.net/Links/2011ITRS/2011Chapters/2011ERD.pdf

[2] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi. Operating system support for NVM+DRAM hybrid main memory. In Proceedings of the 13th Workshop on Hot Topics in Operating Systems , 2009

[3] C. Wang, S. S. Vazhkudai, X. Ma, F. Meng, Y. Kim, C.Engelmann. NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines, In Proceedings of the IEEE International Parallel & Distributed Processing Symposium,2012

[4] R. F. Freitas and W.W.Wilcke. Storage-class memory: The next storage system technology. IBM Journal of Research and Development, 52(4):439–447, 2008.

[5] H.Volos Andres JaanTack, Michael M. S. Mnemosyne: Lightweight Persistent Memory, In Proceedings of the Architectural Support for Programming Languages and Operating Systems ,2011

[6] D.Andersen, J.Franklin, M.Kaminsky, A.Phanishayee, L.Tan, V.Vasudevan :FAWN: A Fast Array of Wimpy Nodes In Proc. 22nd ACM Symposium on Operating Systems Principles, Big Sky, MT. October 2009.

[7] 龙瑞,YAFFS 嵌入式文件系统原理分析[J]. 电脑编程技巧与维护,2006 年第 10 期

[8] F. Douglis, R. Caceres, M. F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, “Storage alternatives for mobile computers,” In Operating Systems Design and Implementation, 1994, pp. 25–37.

[9] D.Woodhouse ,JFFS: The Journalling Flash File System. Ottawa Linux Symposium,2001

[10] C. Manning, “YAFFS: The NAND-specific flash file system,”LinuxDevices.Org, September 2002

[11] A.Hunter. A brief introduction to the design of UBIFS . http://www.linux-mtd.infradead.org/doc/ubifs_whitepaper.pdf.

[12] Memory Technology Devices, http://www.linux-mtd.infradead.org/index.html

[13] UBI - Unsorted Block Images, http://www.linux-mtd.infradead.org/doc/ubi.html

[14] D.Kang ,D.Jung ,J.Kang .et a1． μ-tree:An ordered indexstructure for NAND nash memory, Pfoc of the 7th ACM & IEEE Int Conf on Embedded Software．New York：ACM，2007：144—153

[15] L.M. Grupp, J.D. Davis, S. Swanson. The Bleak Future of NAND Flash Memory. In Proceedings of the USENIX Conference on File and Storage Technologies ,2012

[16] S-H Lee.,K-H Park.An efficient NAND flash file system for flash memory storage in Computers. IEEE Transactions on Computers,2006

[17] C. Min,K Kim,Y.I.Eom et al.SFS :Random Write Considered Harmful in Solid State Drives, In Proceedings of the USENIX Conference on File and Storage Technologies ,2012

[18] T.Pritchett,M.Thottethodi, SieveStore: a highly-selective, ensemble-level disk cache for cost-performance, In Proceedings of the 37th annual international symposium on Computer architecture,2010.

[19] 龙亚春，黄 璞，吴 胜. 超大容量 NAND Flash 文件系统—YAFFS2 在 Linux 下的实现,北京电子科技学院学报，2007,15（2）.

[20] A. Ban. Flash File System, Apr. 1995. United States Patent No. 5,404,485.

[21] Intel. Understanding the Flash Translation Layer (FTL) Specification. Technical report, Intel Corporation, Dec.1998.

[22] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In Proceedings of the Architectural Support for Programming Languages and Operating Systems,2009.

[23] D.Ma,J.Feng,G.Li, LazyFTL: A Page-level Flash Translation Layer Optimized for NAND Flash Memory,In Proceedings of international conference on Management of data,2011.

[24] 郭御风，李琼，刘光明，张磊 基于 NAND 闪存的固态盘技术研究，计算机研究与发展，2009, 46（z2）

[25] A. Ban and R. Hasharon. Flash File System Optimized for Page-mode Flash Technologies, Aug. 1999. United States Patent No. 5,937,425.

[26] S. Choudhuri and T. Givargis. Performance Improvement of Block Based NAND Flash Translation Layer. In Proceedings of International Conference on Hardware/Software Codesign and System Synthesis, 2007.

[27] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. ACM Computing Surveys, 37(2):138–163,June 2005.

[28] J. Kim, J. M. Kim, S. H. Noh, et al. A Space-efficient Flash Translation Layer for CompactFlash Systems. IEEE Transactions on Consumer Electronics, 48(2), May 2002.

[29] S.-W. Lee, D.-J. Park, T.-S. Chung, et al. A Log Buffer Based Flash Translation Layer using Fully Associative Sector Translation. ACM Transactions on Embedded Computing Systems (TECS), 6(3), July 2007.

[30] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In Proceedings of International Conference on Embedded Software,2006.

[31] C. Park, W. Cheon, J. Kang, et al. A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications. ACM Transactions on Embedded Computing Systems, 7(4), July 2008.

[32] S. Lee, D. Shin, Y.-J. Kim, et al. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. ACM The Special Interest Group on Operating Systems, Operating Systems Review, 42(6), Oct. 2008.

[33] H.-S. Lee, H.-S. Yun, and D.-H. Lee. HFTL: Hybrid Flash Translation Layer based on Hot Data Identification for Flash Memory. IEEE Transactions on Consumer Electronics, 55(4), Nov. 2009.

[34] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang. Efficient Identification of Hot Data for Flash Memory Storage Systems. ACM Transactions on Storage, 2(1):22–40, Feb. 2006.

[35] Y.-G. Lee, D. Jung, D. Kang, et al. μ-FTL: A Memory-Efficient Flash Translation Layer Supporting Multiple Mapping Granularities. In Proceedings of International Conference on Embedded Software, 2008.

[36] A.Kawacuchi, S.Nishioka,H.Motoda, .A flash-memory based file system. In Proceedings of the USENIX Technical Conference. New Orleans, LA. 155–164.1995

[37] S. E. Wells ,1994. Method for wear leveling in a flash EEPROM memory. US patent 5,341,339.Filed November 1, 1993; Issued August 23, 1994; Assigned to Intel.

[38] H.-J. Kim ,S.-G. Lee, An effective flash memory manager for reliable flash memoryspace management. IEICE Trans. Inform.Syst. E85-D, 6, 950–964.2002

[39] L.-P. Chang, T.-W. KUO, S.-W. Lo, Realtime garbage collection for flash-memory storage systems of real-time embedded systems. ACM Trans. Embed. Comput. Syst. 3, 4, 837–863 2004

[40] K. Lofgren, R. Norman ,Wear leveIing techniques for flash EEPROM systems：USA，US00735332582[P]．2008

[41] D. Jung, Y-H.Chae,H.Jo,et a1．A group-based wear_leveIing algorithm for large-capacity flash memory storage systems .In Proceedings of Compilers Architecture and Synthesis for Embedded Systems．New York：ACM，2007：160—164

[42] Y. Chang, J. Hsieh, T. Kuo,Endurance enhancement of flash memory storage systems： An efficient static wear leveling design[C] Proc of the 44th AnnuaI conf on Design Automation．New Yorkl ACM，2007：

212—217

[43] Fusion-IO，助力高能耗数据中心节能：整合全球最高性能的存储实现高效节能，http://www.fusioniochina.com/pdf/white_papers/Whitepaper_Green_szh.pdf ，2007

[44] W. K. Josephson, L. A. Bongo , D. Flynn, K. Li, DFS: A file system for virtualized flash storage. In Proceedings of the USENIX Conference on File and Storage Technologies ,2010.

[45] 袁绍龙，专访 Fusion-IO：有固态硬盘还要分层存储干吗？，http://blog.chinaunix.net/space.php?uid=20752346&do=blog&id=59323，2010

[46] J. L .Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufman, CA,1996.

[47] V. S. Pai. A. Badam. SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy. In Proceedings of USENIX Symposium on Networked System Design and Implementation,2011.

[48] M.Saxena and M. M. Swift, FlashVM: Virtual Memory Management on Flash, In USENIX Annual Technical Conference,2010.

[49] D.Jung,J.Kim,S.Park,J.Kang,J.Lee, FASS : A Flash-Aware Swap System, In Proceedings of the International Workshop on Software Support for Portable Storage,2005

[50] S.Ko,S.Jun,Y.Ryu,O.Kwon,K.Koh,A New Linux Swap System for Flash Memory Storage Devices, In Proceedings of the International Conference on Computational Sciences and Its Applications,2008

[51] J.Do,D.Zhang,A.Halerson et al. Turbocharging DBMS buffer pool using SSDs, In Proceedings of the international conference on Management of data,2011

[52] T.Kgil,D.Roberts,T.Mudge, Improving NAND Flash Based Disk Caches, In Proceedings of the International Symposium on Computer Architecture,2008

[53] F.Chen,D.A.Koufaty,X.Zhang, Understanding intrinsic characteristics and system implications of flash memory based solid state drives, In Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems,2009

[54] MXIC: MX30LF1G08AA 1G-bit NAND Flash Memory, http://www.mxic.com.tw/QuickPlace/hq/PageLibrary4825740B00298A3B.nsf/h_Index/8FEA549237D2F7674825795800104C26/$File/MX30LF1G08AA,%203V,%201Gb,%20v0.06.pdf, 2012

[55] MXIC: MX68GL1G0F H/L, http://www.mxic.com.tw/QuickPlace/hq/PageLibrary4825740B00298A3B.nsf/h_Index/CDFF7817E4C5F42148257961000DC5BF/$File/MX68GL1G0F%20H-L,%203V,%201Gb,%20v0.00.pdf, 2011

[56] 郑文静、李明强、舒继武. Flash 存储技术，计算机研究与发展，47（4）：716-726，2010

作者简介：

魏　巍： 中科院计算所先进计算机系统实验室系统结构组硕士研究生

熊　劲： 中科院计算所先进计算机系统实验室系统结构组副研究员 xiongjin@ict.ac.cn

蒋德钧： 中科院计算所先进计算机系统实验室系统结构组助理研究员